计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(2):35−42 [doi: 10.15888/j.cnki.csa.007769]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于奇异值分解的图像压缩技术①

邓子云

(长沙商贸旅游职业技术学院 经济贸易学院, 长沙 410116)
通讯作者: 邓子云, E-mail: dengziyun@126.com

摘　要: 为获得比较理想的图像压缩比和清晰的压缩后图像, 使用了奇异值分解作为数据矩阵的压缩原理. 详细解析了奇异值分解的原理及用奇异值分解压缩图像的原理. 提出了按特征值个数占比阈值、按特征值之和占比阈值两种取特征值个数的方法. 实验表明, 特征值个数占比阈值在 0.1 时, 图像清晰且压缩比达到 5.99; 特征值之和占比阈值在 0.85 时, 图像清晰, 对 PNG 格式图像压缩比达到 7.89, 对 JPG 格式图像压缩比达到 5.92. 从实验的个例来看, 前 1% 的特征值表征了较多份量的数据特征, 在征值个数占比阈值确定时, 对 PNG 格式和对 JPG 格式图像压缩比相同; 在特征值之和占比阈值确定时, 对 PNG 格式相对比对 JPG 格式图像压缩比要高. 认为按特征值之和占比阈值取特征值个数这种压缩方法更具普适性, 可适用于 Alpha 通道冗余的情况, 并可为大规模数量的图像压缩设定统一的特征值之和占比阈值.

关键词: 奇异值分解; 图像压缩; 占比阈值; 特征值; 压缩比

引用格式: 邓子云.基于奇异值分解的图像压缩技术.计算机系统应用,2021,30(2):35–42. http://www.c-s-a.org.cn/1003-3254/7769.html

Image Compression Based on Singular Value Decomposition

DENG Zi-Yun

(Faculty of Economics and Trade, Changsha Commerce & Tourism College, Changsha 410116, China)

Abstract: Singular Value Decomposition (SVD) is adopted for image compression of the data matrix to obtain an optimal compression ratio and a clear compressed image. The principle of SVD and its application to compressing images are elaborated. Two methods for obtaining the better number of eigenvalues are proposed including the ratio threshold of eigenvalue number and the ratio threshold of eigenvalue sum. The experiments reveal that when the ratio threshold of eigenvalue number is 0.1, a clear image is obtained with the compression ratio of 5.99. When the ratio threshold of eigenvalue sum is 0.85, a clear image is also acquired with the compression ratio for PNG images of 7.89 and that for JPG images of 5.92. Case study indicates that the first 1% of eigenvalues represent more data characteristics. When the ratio threshold of eigenvalue number is determined, the compression ratios for PNG and JPG images are identical. When the ratio threshold of the eigenvalue sum is determined, the compression ratio for PNG images is higher than that for JPG images. The method for obtaining the eigenvalue number according to the ratio threshold of eigenvalue sum is more universal. It can be applied to solving alpha channel redundancy and setting a unified ratio threshold of eigenvalue sum for large-scale image compression.

Key words: Singular Value Decomposition (SVD); image compression; ratio threshold; eigenvalue; compression ratio

① 基金项目: 湖南省自然科学基金 (2020JJ7091); 中国物流学会研究课题 (2019CSLKT3-226)
Foundation item: Natural Science Foundation of Hunan Province (2020JJ7091); Research Project of China Society of Logistics (2019CSLKT3-226)
收稿时间: 2020-06-08; 修改时间: 2020-07-07; 采用时间: 2020-07-17; csa 在线出版时间: 2021-01-27

图像压缩是计算机学科学术界和工程界关注的热门研究领域之一[1,2]. 目前已有很多成熟的压缩算法, 也产生了各种图像压缩格式, 如 JPG、GIF 等[3]. 还有一些通用的文件压缩算法, 进而产生了各种文件压缩格式, 如 ZIP、RAR 等[4]. 奇异值分解方法是线性代数数学学科中一种数据压缩方法, 可以将大规模的矩阵在分解为矩阵的乘法表示后, 用一定比例的特征数据矩阵来表示原来的大规模矩阵, 从而达到压缩的目的. 奇异值分解方法可以运用到图像压缩领域, 并起到良好的压缩效果.

已有一些学者、工程技术人员将奇异值分解方法运用到特定的图像压缩应用上, 如视频监控图像、遥感图像等[5,6]. 还有的研究人员结合奇异值分解方法和其它的算法来提升图像的压缩比[7], 绝大多数研究工作采用的是 Matlab 开发工具[8,9]. 考虑到图像压缩算法多已成熟, 本文并不打算提出新的算法, 而是深入研究基于奇异值分解的技术原理, 提出 2 种运用奇异值分解作图像压缩的方法, 用 Python 编程实现并展开实验, 再对 JPG、PNG 这 2 种通用的图像格式作出压缩效果的示例和对比分析.

1 奇异值分解的原理

奇异值分解可以针对任意形态的矩阵作特征值分解. 现实应用场景中的数据确实不太可能都是方阵, 而多是行数、列数不等的数据矩阵, 因此, 奇异值分解具有广泛的应用价值[10].

奇异值分解的通用形式如式 (1) 所示[11−13]:

$$A_{m\times n} = U_{m\times m} D_{m\times n} V^{-1}_{n\times n} = U_{m\times m} D_{m\times n} V^{\mathrm{T}}_{n\times n} \tag{1}$$

式 (1) 中的$U_{m\times m}$被称为左奇异向量组成的标准正交基矩阵, $D_{m\times n}$被称为特征值对角矩阵, $V^{\mathrm{T}}_{n\times n}$被称为右奇异向量组成的标准正交基矩阵. 不论是$m > n$, 还是$m \le n$, 式 (1) 都会成立. 式 (1) 可用更为形象的图形来表述, 如图 1(a) 和图 1(b) 所示[14].

2 用奇异值分解压缩图像的原理

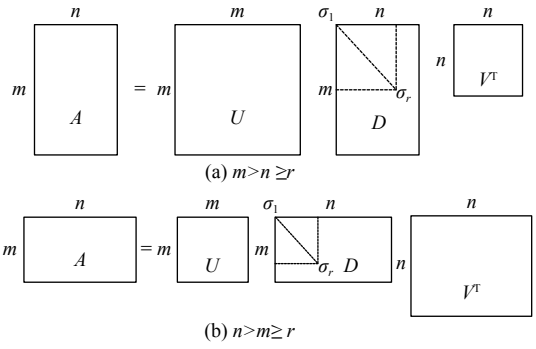
根据式 (1), 如果将 r 个特征值从大到小排序, 并调动对应的$U_{m\times m}$和$V^{\mathrm{T}}_{n\times n}$中的向量位置, 可以得到数据矩阵 A 的奇异值分解排序后的结果, 这个分解结果即可用于压缩数据矩阵 A.

2.1 如何压缩矩阵数据

具体压缩方法是取前 k 个特征值, 及矩阵$U_{m\times m}$和矩阵$V^{\mathrm{T}}_{n\times n}$中的前 k 个向量, 可得到:

$$A \approx \sigma_1 u_1 V_1^{\mathrm{T}} + \sigma_2 u_2 V_2^{\mathrm{T}} + \cdots + \sigma_k u_k V_k^{\mathrm{T}} \tag{2}$$

以图 1(a) 为例, 这种数据压缩的思想示意如图 2 所示.



(a) $m>n\ge r$

(b) $n>m\ge r$

图 1 奇异值分解的图示

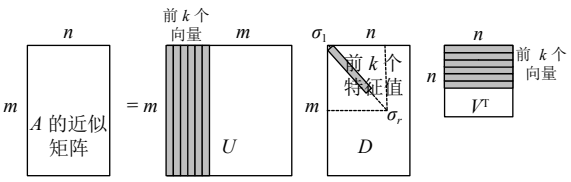


图 2 数据压缩的思想图示

这种数据压缩的思想体现的就是用数据的主要成份来代表整体数据, 从而实现只要存储较少的数据, 可见这其实是一种有损数据压缩, 因此需要将数据的压缩控制在可以接受的范围[15].

2.2 压缩比的计算

经过如图 2 所示的变换后, 数据矩阵 A 的近似矩阵的行数和列数并没有变, 那又怎么是节约空间了呢?

这是因为奇异值分解后, 不必再存储数据矩阵 A, 而是存储矩阵$U_{m\times m}$的前 k 列、矩阵$D_{m\times n}$中的前 k 个特征值, 矩阵$V^{\mathrm{T}}_{n\times n}$的前 k 行, 通过式 (2) 计算再得到数据矩阵 A 的近似矩阵. 这样, 要存储的数据远比存储数据矩阵 A 要使用的存储空间少得多.

以一个 1000×2000 的数据矩阵为例, 则要存储 2 000 000 个数据, 假定每个数据占用的存储空间数量相同. 假定运用奇异值分解共需使用 50 个特征值来作数据压缩, 存储矩阵$U_{m\times m}$的前 50 列需存储$1000\times 50 = 50\,000$个数据, 存储矩阵$V^{\mathrm{T}}_{n\times n}$的前 k 行需存储$2000\times 50 = 100\,000$个数据, 则总计需要存储$50\,000 + 100\,000 + 50 = 150\,050$个数据, 这远比存储 2 000 000 个数据要节约存储空间.

衡量这种节约的程度可用压缩比来表示, 按式 (3) 所示的公式计算:

$$P = \frac{S_{原}}{S_{压}} \tag{3}$$

式 (3) 中, P 表示压缩比, $S_{原}$表示原始数据占用的存储空间, $S_{压}$表示压缩后的数据占用的存储空间. 因此在上述例子中:

$$P = \frac{2\,000\,000}{150\,050} \approx 13.33$$

2.3 k 取值的 2 种方法

k 取值应为多少合适呢?那得看对数据压缩的目标需求了, k 越大, 数据就压缩得越少, 需要的存储空间也越多, 需要找到一个合适的平衡点. 有 2 种方法:

(1) 按特征值个数占比阈值取特征值个数. 设定一个比例阈值 f, 如果 k 与特征值总个数之比的值超过阈值 f, 则取前 k 个特征值.

(2) 按特征值之和占比阈值取特征值个数. 设定一个比例阈值 f, 如果前 k 个特征值之和与所有特征值之和的比的值超过阈值 f, 则取前 k 个特征值.

2.4 图像压缩的原理

以常用的 PNG 和 JPG 格式的图像为例, 读取它们的图像数据可得到一个 3 维的数据矩阵. 第 1 维表示的横向的像素, 第 2 维表示纵向的像素, 第 3 维表示图像的通道, 用 0~255 范围的数据表示数据值.

PNG 和 JPG 两种格式不同的是, PNG 格式图像的第 3 维有 4 个通道, 分别表示 R (Red, 红色)、G (Green, 绿色)、B (Blue, 蓝色)、A (Alpha, 透明度); 而 JPG 格式图像只有 R、G、B 这 3 个通道, 没有 A 这个通道[16]. 这也是 JPG 格式图像比 PNG 格式图像占用空间更小的根本原因. 此外, 两种格式对数据均有压缩的算法. 这里不讨论并忽略两种格式本身的数据压缩算法.

在分离得到 PNG 格式图像的 R、G、B、A 这 4 个通道的数据矩阵后, 可对这 4 个 2 维数据矩阵分别作奇异值分解, 再根据使用的 k 取值的方法和阈值 f, 可得到这 4 个数据矩阵的前 k 个特征值、$U_{m\times k}$和$V^{\mathrm{T}}_{k\times n}$. JPG 格式图像则不需要对 A 通道的数据矩阵作奇异值分解.

要查看压缩后的 PNG 格式图像, 则可将 4 个通道的前 k 个特征值、$U_{m\times k}$和$V^{\mathrm{T}}_{k\times n}$根据式 (2) 分别计算得到近似的数据矩阵, 组合这 4 个 2 维数据矩阵形成 PNG 格式图像的 3 维数据, 即可显示图像. JPG 格式图像则计算并组合 R、G、B 这 3 个通道的压缩后数据得到图像的 3 维数据.

3 图像压缩应用示例

这里以一张原图的 PNG、JPG 2 种格式为例, 用 k 取值的 2 种方法来展开图像压缩实验.

3.1 图像原图

使用的图像原图如图 3 所示.

图 3 原图

原图宽度为 4928 像素, 高度为 3264 像素, 用 8 位整数表示各通道的值. 则 PNG 原图占用的存储空间为 (不考虑 PNG 格式本身的压缩效果):

$$4928\times3264\times4\times8 = 514\,719\,744 \text{ bits} \approx 61.36 \text{ MB}$$

JPG 原图占用的存储空间为 (不考虑 JPG 格式本身的压缩效果):

$$4928\times3264\times3\times8 = 386\,039\,808 \text{ bits} \approx 46.02 \text{ MB}$$

在 Python 中, 可用代码 1 的源代码获取 PNG 原图 4 个通道数据矩阵.

代码 1. 获取 PNG 原图 4 通道数据矩阵

```
import numpy as np
from PIL import Image
#加载图像, 第 1 个参数为原图的完整路径
orignImage=Image.open(r'原图.png','r')
imageArray=np.array(orignImage)
#得到 R 通道数据矩阵
R=imageArray[:,:,0]
#得到 G 通道数据矩阵
G=imageArray[:,:,1]
#得到 B 通道数据矩阵
B=imageArray[:,:,2]
#得到 A 通道数据矩阵
#原图为 JPG 时应注释此句源代码
A=imageArray[:,:,3]
```

PIL 为 Python 的一个第三方图像处理类库, 事先应在操作系统的命令界面用语句“pip install pillow”来安装.

3.2 按特征值个数占比阈值取特征值个数

在 Python 中, 用代码 2 即可得到一个数据矩阵的奇异值分解结果.

代码 2. 数据矩阵奇异值分解

```
import numpy as np
#对 R 通道数据矩阵作奇异值分解
U_R,sigma_R,V_T_R=np.linalg.svd(R)
#对 G 通道数据矩阵作奇异值分解
U_G,sigma_G,V_T_G=np.linalg.svd(G)
#对 B 通道数据矩阵作奇异值分解
U_B,sigma_B,V_T_B=np.linalg.svd(B)
#对 A 通道数据矩阵作奇异值分解
#为 JPG 原图时应注释此句源代码
U_A,sigma_A,V_T_A=np.linalg.svd(A)
```

可以发现, 在作奇异值分解后, sigma_R、sigma_G、sigma_B、sigma_A 均为一维数组, 其元素个数均为 3264, 则表明均有 3264 个特征值. 需要注意的是, Python 中的 0 会表示为一个很小的值, 而不会表示为整型的 0, 因此有的通道数据矩阵可能并没有 3264 个特征值, 需要编程判断, 可以用特征值与一个很小的值 (如 0.0001) 比较, 如果小于这个很小的值, 则将特征时判断为 0. 结果发现, sigma_A 的特征值只有 1 个. 为什么会这样呢?说明 A 通道数据是冗余的.

设计一个函数, 用于生成指定的比例阈值下, 用通道的压缩后数据来生成图像的近似数据矩阵, 如代码 3 所示.

代码 3. 指定比例阈值下的图像近似数据矩阵生成

```
#功能: 针对某个通道的数据矩阵作奇异值分解得到的 U 矩阵、sigma
#数组、V_T 矩阵, 根据 percent(百分比) 取前若干个特征值来生成
#该通道的近似数据矩阵
#参数: U, 对某个通道的数据作矩阵奇异值分解后得到的 U 矩阵; sigma,
#对某个通道的数据作矩阵奇异值分解后得到的 sigma 数组; V_T, 对
#某个通道的数据作矩阵奇异值分解后得到的 V_T 矩阵; percent, 特
#征值个数占比阈值.
#返回值: 图像的某个通道的近似数据矩阵
def genCompressData(U,sigma,V_T,percent):
    m=U.shape[0]
    n=V_T.shape[0]
    reChannel=np.zeros((m,n))
    for k in range(len(sigma)):
    #以得到该通道的近似数据矩阵
    #逐个累加
    reChannel=reChannel+sigma[k]*np.dot(U[:,k].reshape(m,1),
    V_T[k,:].reshape(1,n))
    #如果已经超过设定的比例阈值
    if (float(k)/len(sigma)>percent):
        #将数据值规范到 0-255 范围内
        reChannel[reChannel<0]=0
        reChannel[reChannel>255]=255
        break
#将返回的近似数据矩阵的元素数据类型规范
#为 uint8
return np.rint(reChannel).astype("uint8")
```

取特征值个数占比阈值 f 分别为 0.001、0.005、0.01、0.02、0.03、0.04、0.05、0.1. 可用代码 4 逐个形成并保存压缩后再生成的图像.

代码 4. 形成并保存过程

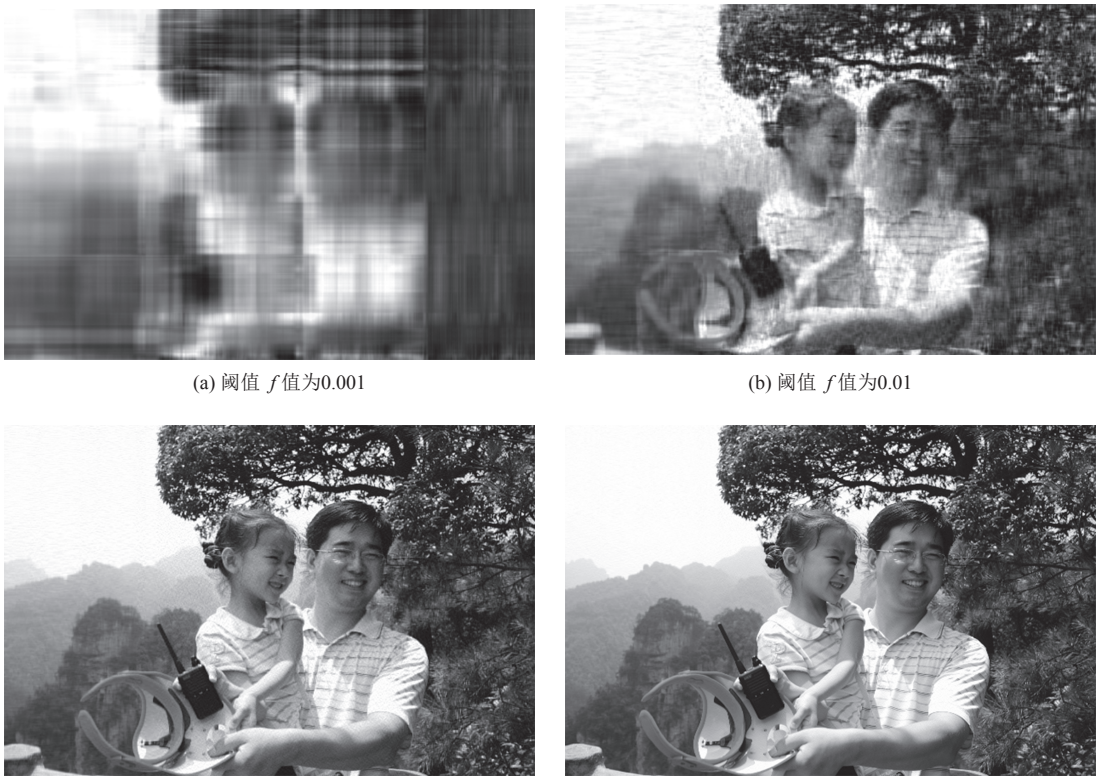
```
for p in [0.001,0.005,0.01,0.02,0.03,0.04,_0.05,0.1]:
  #生成 R 通道近似数据矩阵
  reR=genCompressData(U_R,sigma_R,V_T_R,p)
  #生成 G 通道近似数据矩阵
  reG=genCompressData(U_G,sigma_G,V_T_G,p)
  #生成 B 通道近似数据矩阵
  reB=genCompressData(U_B,sigma_B,V_T_B,p)
  #生成 A 通道近似数据矩阵, 为 JPG 原图时应注释此句源代码
  reA=genCompressData(U_A,sigma_A,V_T_A,p)
  #生成完整的近似数据 3 维矩阵, 原图为 JPG 时则转而使用下面的
  #注释语句
  reI=np.stack((reR,reG,reB,reA),2)
  # reI=np.stack((reR,reG,reB),2)
  reI=np.stack((reR,reG,reB,reA),2)
  Image.fromarray(reI).save
    ("{}".format(p)+"img.png")
```

为简便起见, 取比例阈值 f 值为 0.001、0.01、0.05、0.1 时的 4 幅压缩效果图作出展示, 如图 4 所示. 从图中可以看出, 比例阈值 f 值为 0.001 图不清楚, 为 0.01 值时可以大致看出轮廓, 为 0.05 时图像基本可辨, 为 0.1 时图像与原图相差无几, 人眼分辨不出是否压缩过.

对 JPG 格式图像的实验结果这里不再重复罗列分析. 在计算出取不同的比例阈值下的压缩比后, 列出结果如表 1 所示.

从表 1 可以看到, 即使是取比例阈值 f 为 0.1, 压缩比都还能达到 5.99, 因此, 压缩效果良好. 如果对压缩后的图像清晰度要求可以降低, 则还可以得到更高的压缩比.

表 1 中的两种格式的图像的压缩比相同的原因是, 不管是 3 个通道还是 4 个通道, 在同一比例阈值下, 针对各个通道取特征值的个数相同, 故压缩比就会相同.



(a) 阈值 f 值为0.001　(b) 阈值 f 值为0.01

(c) 阈值 f 值为0.05　(d) 阈值 f 值为0.1

图 4　按特征值个数占比阈值取特征值个数时的压缩效果图

表 1　按特征值个数占比阈值取特征值个数时的压缩比 (4 个通道均相同)

比例阈值f	特征个数k	PNG 存储空间(MB)	PNG 压缩比	JPG 存储空间(MB)	JPG 压缩比
0.001	5	0.16	392.65	0.12	392.65
0.005	18	0.56	109.07	0.42	109.07
0.01	34	1.06	57.74	0.8	57.74
0.02	67	2.09	29.3	1.57	29.3
0.03	99	3.09	19.83	2.32	19.83
0.04	132	4.13	14.87	3.09	14.87
0.05	165	5.16	11.9	3.87	11.9
0.1	328	10.25	5.99	7.69	5.99

注: (1) PNG原图占用的存储空间为61.36 MB;
(2) JPG原图占用的存储空间为46.302 MB.

3.3　按特征值之和占比阈值取特征值个数

设计一个函数 (代码 5), 用于生成指定的比例阈值下, 用各通道的压缩后数据来生成图像的近似数据矩阵.

代码 5. 图像近似数据矩阵生成

```
#功能: 针对某个通道的数据矩阵作奇异值分解得到的 U 矩阵、sigma
#数组、V_T 矩阵, 根据 percent(百分比) 取前若干个特征值来生成
#图像的该通道的近似数据矩阵;
#参数: U, 对某个通道的数据做矩阵奇异值分解后得到的 U 矩阵;
#sigma, 对某个通道的数据作矩阵奇异值分解后得到的 sigma 数组;
#V_T, 对某个通道的数据作矩阵奇异值分解后得到的 V_T 矩阵;
#percent, 特征值之和占比阈值.
#返回值: 图像的该通道的近似数据矩阵
def genCompressDataFromSum(U,sigma,V_T,percent):
    m=U.shape[0]
    n=V_T.shape[0]
    reChannel=np.zeros((m,n))
    sum=0.0 #sum 为特征值总和
    for i in sigma:
        sum+=i
    # sumcurrent 为已累加的特征值的和
    sumcurrent=0.0
    for k in range(len(sigma)):
        #逐个累加, 以得到该通道的近似数据矩阵
        reChannel=reChannel+sigma[k]*np.dot
            (U[:,k].reshape(m,1),V_T[k,:].reshape(1,n))
        #累加特征值
        sumcurrent+=sigma[k]
        #如果已经超过设定的比例阈值
        if (sumcurrent/sum>percent):
            #将数据值规范到 0-255 范围内
            reChannel[reChannel<0]=0
            reChannel[reChannel>255]=255
            break
    #将返回的近似数据矩阵的元素数据类型规范为 uint8
    return np.rint(reChannel).astype("uint8")
```

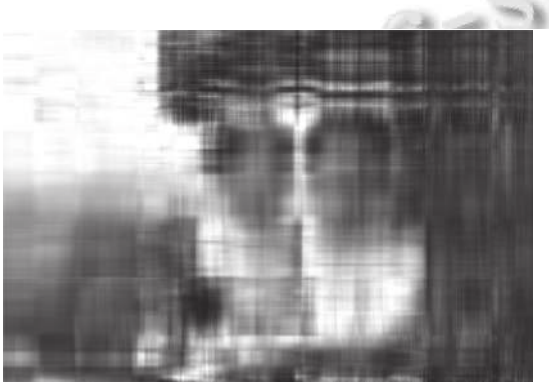
取特征值之和占比阈值 f 分别为 0.5、0.6、0.7、0.8、0.9, 可用代码 6 逐个形成并保存压缩后再生成的图像.

代码 6. 形成并保存过程

```
for p in[0.3,0.4,0.5,06,0.7,0.8,0.85,0.9]:
  #生成 R 通道近似数据矩阵
  reR= genCompressDataFromSum(U_R,sigma_R,V_T_R,p)
  #生成 G 通道近似数据矩阵
  reG= genCompressDataFromSum(U_G,sigma_G,V_T_G,p)
  #生成 B 通道近似数据矩阵
  reB= genCompressDataFromSum(U_B,sigma_B,V_T_B,p)
  #生成 A 通道近似数据矩阵, 为原图 JPG 时应注释此句源代码
  reA= genCompressDataFromSum(U_A,sigma_A,V_T_A,p)
  #生成完整的近似数据 3 维矩阵, 原图为 JPG 时则转而使用下面的
  #注释语句
  reI=np.stack((reR,reG,reB,reA),2)
  # reI=np.stack((reR,reG,reB),2)
  #保存图像, 参数为图像的完整路径
  Image.fromarray(reI).save
    ("{}".format(p)+"img.png")
```

为简便起见, 取比例阈值 f 值为 0.6、0.7、0.8、0.85 时的 4 幅压缩效果图作出展示, 如图 5 所示. 可以看到, 当比例阈值 f 值为 0.7 时, 已基本可辨; 当比例阈值 f 值为 0.85 时, 图片已经比较清晰. 表 2 还给出了取不同的阈值 f 值时, 对应的各个通道的特征值个数, 以及针对 PNG 格式图像、针对 JPG 格式图像的存储空间和压缩比.

表 2 中 PNG 格式图像的存储空间和 JPG 格式图像的存储空间相同, 且 A 通道的 k 值为 1 的原因是, PNG 格式图像原图的 A 通道中的值都是等值的 255, 相当于这个通道的数据是冗余的, 其图片显示效果与 JPG 格式图像相当.



(a) 阈值 f 值为0.3

(b) 阈值 f 值为0.5

(c) 阈值 f 值为0.7

(d) 阈值 f 值为0.85

图 5 按特征值之和占比阈值取特征值个数时的压缩效果图

3.4 压缩比变化趋势分析

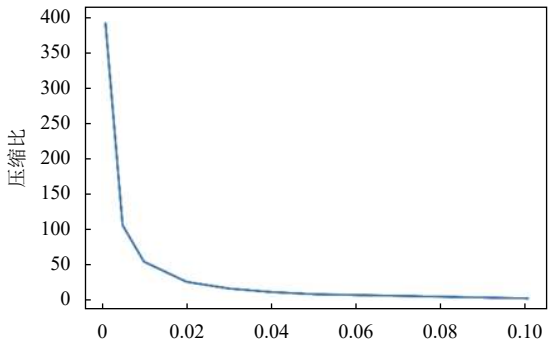
根据表 1 和表 2, 作比例阈值和压缩比之间的关系图, 如图 6 所示.

从图 6(a) 来看, 对 PNG 格式图像的压缩比和对 JPG 格式图像的压缩比的变化曲线相同. 在比例阈值为 0.01 时出现曲线拐点, 这表明在前 1% 的特征值里, 较大值的特征值比较集中; 之后曲线变缓, 在比例阈值为 0.1 以后, 曲线已经非常平缓, 表明增加特征值已很难再改善图像质量.

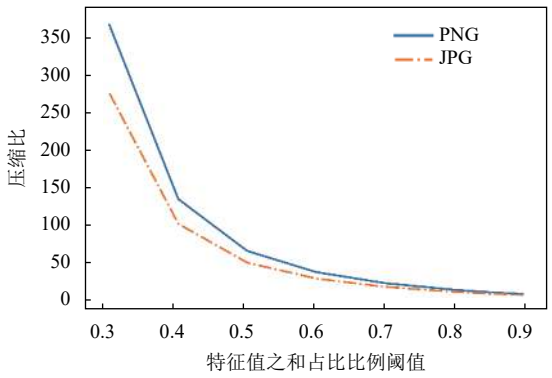
从图 6(b) 来看, 对 JPG 格式图像的压缩比要比对 PNG 格式图像的压缩比低一点. 在比例阈值为 0.9 后, 曲线变得更为平缓, 这表明增加特征值已很难再改善图像质量.

表 2 按特征值之和占比阈值取特征值个数时的压缩比 (通道 R, G, B, A)

比例阈值f	特征个数k	PNG 存储空间(MB)	PNG 压缩比	JPG 存储空间(MB)	JPG 压缩比
0.3	7,7,7,1	0.16	373.95	0.16	280.47
0.4	20,19,19,1	0.45	135.4	0.45	101.55
0.5	41,40,38,1	0.95	64.37	0.95	48.28
0.6	76,73,70,1	1.73	35.37	1.73	26.53
0.7	132,129,124,1	3.03	20.24	3.03	15.18
0.8	235,234,231,1	5.49	11.17	5.49	8.38
0.85	329,331,332,1	7.77	7.89	7.77	5.92
0.9	494,499,508,1	11.75	5.22	11.75	3.92



(a) 按特征值个数占比阈值取特征值个数



(b) 按特征值之和占比阈值取特征值个数

图 6 比例阈值和压缩比之间的关系图

相对来说, 按特征值之和占比阈值取特征值个数的方法更为实用. 因为首先, 它考虑的是特征值的重要程度, 而不是个数; 其次这种方法可以应对个别通道数据冗余的问题. 更为重要的是, 这种方法可以用于进行大规模数量的图像文件压缩, 因为这种方法可以划定一个统一的可以接受的特征值之和占比阈值, 这个阈值就直接代表着图像的清晰度. 鉴于前述对占比阈值的分析, 认为这个统一的占比阈值如要基本可辨取 0.7, 如要比较清晰取 0.85. 但按特征值个数占比阈值取特征值个数的方法不能划定一个统一的比例阈值, 因为在同一个阈值下, 不同的图像文件的清晰程度可能不同.

3.5 压缩算法对比分析

现已有很多经典的图像压缩算法. 以经典的 PNG[5]、JPG[6] 图像格式为对比, 仍以本文的图片作为示例, 与本文的压缩方法在压缩比上的对比分析如表 3 所示. 由此可见, 在对 PNG 按特征值之和占比阈值取特征值个数 (f=0.7) 时可以取得最高的压缩比.

表 3 对不同图像格式处理算法的压缩比对比

图像格式	算法	存储空间(MB)	压缩比(%)
PNG	经典算法	19.42	3.16
	本文f=0.7	3.03	20.24
	本文f=0.85	7.77	7.89
JPG	经典算法	5.83	7.94
	本文f=0.7	3.03	15.18
	本文f=0.85	7.77	5.92

鉴于各种算法的原理不同, Python 已经提供了成熟的软件包 PIL, 直接用保存功能即可将图片存储为 JPG 或 PNG 格式, 不便计算时间效率. 而本文给出的算法关键之处在于做奇异值分解所耗费的时间, 仍以本文的图片作为示例, 实验结果如表 4 所示. 可见, 对 JPG 做奇异值分解效率明显较高, 但显然处理一个图片已超过 1 分钟, 效率有待提升.

表 4 奇异值分解时间效率

分类	消耗时间(s)
PNG	308.90
JPG	78.61

4 结束语

通过奇异值分解, 可以得到 3 个分解结果, 即左奇异向量组成的标准正交基矩阵$U_{m\times m}$、特征值对角矩阵$D_{m\times n}$、右奇异向量组成的标准正交基矩阵$V^{\mathrm{T}}_{n\times n}$. 有按特征值个数占比阈值取特征值个数和按特征值之和占比阈值取特征值个数 2 种方法来做图像压缩, 从而得到$D_{m\times n}$中的前 k 个特征值、$U_{m\times m}$的前 k 列、$V^{\mathrm{T}}_{n\times n}$的前 k 行作为要存储的数据来代表数据矩阵. 通过对 PNG、JPG 两种格式图像作压缩实验发现, 在本实验个例中, 特征值个数占比阈值取 0.1 时, 图片清晰, 压缩比达到 5.99; 特征值之和占比阈值取 0.85 时, 图片清晰, 对 PNG 格式图像的压缩比达到 7.89, 对 JPG 格式图像的压缩比达到 5.92. 认为相对来说, 按特征值之和

占比阈值取特征值个数的做法更为实用, 可用于大规模数量的图像文件压缩.

本文研究的局限性在于仅限于奇异值分解本身并应用于图像压缩领域, 没有将奇异值分解和其它压缩算法相结合开展研究. 下一步的研究可结合包括奇异值分解在内的多种压缩算法提出新的组合算法, 并作大规模数量的图像文件处理实验、对比分析, 以寻求具有更高压缩比、能普适应用的图像压缩方法.

参考文献

1 Mou J, Yang FF, Chu R, *et al*. Image compression and encryption algorithm based on hyper-chaotic map. Mobile Networks and Applications, 2019. [doi: 10.1007/s11036-019-01293-9]

2 王棒. 基于压缩感知和极余弦变换的图像多功能双水印算法研究 [硕士学位论文]. 西安: 西安理工大学, 2019.

3 Blasch E, Chen HM, Irvine JM, *et al*. Prediction of compression-induced image interpretability degradation. Optical Engineering, 2018, 57(4): 043108.

4 Shapira D, Storer JA. In place differential file compression. The Computer Journal, 2005, 48(6): 677–691. [doi: 10.1093/comjnl/bxh128]

5 胡启杨. 基于压缩感知的视频监控图像处理算法研究 [硕士学位论文]. 北京: 华北电力大学, 2018.

6 霍英海. 基于压缩感知的遥感影像融合技术研究 [硕士学位论文]. 重庆: 重庆邮电大学, 2019.

7 Epps BP, Krivitzky EM. Singular value decomposition of noisy data: Mode corruption. Experiments in Fluids, 2019, 60(8): 121. [doi: 10.1007/s00348-019-2761-y]

8 Allanki S, Dixit M, Thangaraj P, *et al*. Analysis and modelling of septic shock microarray data using Singular value decomposition. Journal of Biomedical Informatics, 2017, 70: 77–84. [doi: 10.1016/j.jbi.2017.05.005]

9 徐小敏. 超宽带穿墙雷达杂波抑制与成像方法研究 [硕士学位论文]. 南京: 南京信息工程大学, 2019.

10 郭明军, 李伟光, 杨期江, 等. 基于 SVD 原理的 PCA 特征频率提取算法及其应用. 华南理工大学学报 (自然科学版), 2020, 48(1): 1–9.

11 Son D, Park Y. Generalization of head-related transfer function database using tensor-singular value decomposition. Noise Control Engineering Journal, 2017, 65(5): 454–461. [doi: 10.3397/1/376561]

12 张啸剑, 付聪聪, 孟小峰. 结合矩阵分解与差分隐私的人脸图像发布. 中国图象图形学报, 2020, 25(4): 655–668. [doi: 10.11834/jig.190308]

13 马敏, 何小芳, 李明, 等. 基于改进 TSVD 正则化的 ECT 图像重建算法. 传感器与微系统, 2020, 39(4): 136–139.

14 杨智伟, 刘灏, 毕天姝, 等. 基于奇异值分解的 PMU 数据恢复法. 中国电机工程学报, 2020, 40(3): 812–821.

15 Andono PN, Supriyanto C, Nugroho S. Image compression based on SVD for BoVW model in fingerprint classification. Journal of Intelligent & Fuzzy Systems, 2018, 34(4): 2513–2519.

16 王俊. 基于 SVD 裁剪的深度神经网络压缩技术研究与实现 [硕士学位论文]. 北京: 北京邮电大学, 2019.